Podding Manual

1. 환경 설정

실행 환경 및 사용 툴

1) 실행 환경

1. Windows

각종 IDE를 통한 작업과 문서 작성을 위한 개발환경은 Windows 10 Enterprise Version 22H2 위에서 이루어졌다. Notion, Jira, IntelliJ, VSCode, Git 등의 협업과 개발을 위한 툴을 Window 버전으로 사용하였다.

2. Ubuntu

배포환경에 사용한 AWS EC2 인스턴스 이미지는 Ubuntu 20.04 Focal 을 사용하였다.

3. Docker

Spring Boot, Nginx, MySQL, Jenkins 어플리케이션을 담은 컨테이너는 다음과 같은 기본 이미지에서 빌드하였다.

어플리케이션	이미지	비고
Spring Boot	ubuntu:22.04	Dockerfile 빌드
Nginx	ubuntu:22.04	Dockerfile 빌드
MySQL	mysql:8.0.32-debian	
Jenkins	jenkins:lts-jdk11	Dockerfile 빌드

표 1. 어플리케이션 별 사용 이미지

4. Java

Spring Boot JAR 아카이브 실행을 위해 JRE를 포함하는 JDK는 Zulu JDK를 사용하였다. Zulu-8.68.0.21-OpenJDK-8.0.362 버전을 JRE가 필요한 Spring Boot 컨테이너에 설치하여 개발와 배포를 진행하였다.

이와는 별개로, Jenkins의 컨테이너의 이미지는 JDK-11 버전을 제공한다.

5. NodeJS

Vite 빌드를 위해 NodeJS 19.6.0 버전과 NPM 9.4.0 버전을 Jenkins 컨테이너에 설치하였다.

2) 툴

1. IDE

개발에 사용한 IDE와 버전정보는 다음과 같다.

용도	버전	빌드
Spring Boot 개발	JetBrains IntelliJ IDEA 2022.3.1	#IU-223.8214.52
React 개발(정)	Microsoft Visual Studio Code 1.75.1	
React 개발(부)	JetBrains WebStorm 2022.3.1	#WS-223.8214.51
DB 관리(정)	MySQL Workbench 8.0.31 Community	2235049 CE
DB 관리(부)	JetBrains DataGrip 2022.3.2	#DB-223.8214.62

표 2. 용도 별 사용 IDE

2. Spring Boot

Spring Boot 2.7.8 버전과 Spring Boot Data, Spring Security, Spring Boot Web 등 각종 의존성을 추가하여 개발하였다. 본 프로젝트 내의 pom.xml 파일을 참고하여 추가한 의존성을 확인할 수 있다.

3. Maven

의존성 관리를 위해 Maven 4.0.0 버전을 사용하였다.

4. Nginx

Vite 빌드 후 생성된 정적파일을 제공하기 위해 Nginx 1.18.0 버전의 서버를 사용하였다. 프로젝트 내의 myapp.conf 설정파일을 통해 세부 설정을 확인할 수 있다.

5. React

웹페이지 개발을 위해 React 18.2.0 버전을 사용하였다. package.json 파일을 통해 세부 버전을 확인할 수 있다.

6. Vite

React 앱의 빠른 빌드를 위해 Vite 4.1.1 버전을 사용하였다. package-lock.json 파일을 통해 세부 버전을 확인할 수 있다.

7. MySQL

로컬 개발환경의 데이터베이스는 MySQL Server 8.0.31 버전을 사용하였다. 실제 배포 MySQL 환경의 데이터베이스는 MySQL Server 8.0.32 버전을 사용하였다.

8. Jenkins

GitLab과 연동을 통한 자동배포는 Jenkins 를 통해 이루어졌다. Jenkins 2.375.2 버전을

사용하여 이를 진행하였다.

9. **Docker Compose**

Jenkin 를 통해 컨테이너 오케스트레이션을 진행하기 위해서 docker-compose를 설치하였다. Docker Compose 2.15.1 버전을 사용하였다.

환경변수

1) Development

1. **Backend**

spring.profiles.active=dev

배포환경과 개발환경 분리를 위해 Spring Boot의 application.yml 프로필을 분리하였다. java –jar -Dspring.profiles.active=dev 옵션을 통해 프로필을 활성화시킬 수 있다.

2. **Frontend**

아래의 환경변수들은 Vite 빌드 시 자동으로 추가되므로, 환경변수를 따로 추가할 필요는 없다. 프로젝트 내의 .env 파일에서 확인 가능하다.

VITE_APP_REST_API_KEY="98268e53473ceb3e11dd6e609a5fa990"

Kakao OAuth 인증을 위해 필요한 API 시크릿 키이다.

VITE_APP_IMAGE_ROOT=https://storage.googleapis.com/studymoim/

사진 파일들을 다운로드, 업로드하기 위해 접근해야하는 클라우드 파일시스템 경로이다.

VITE_APP_API_SERVER="localhost:8080"

Spring Boot 서버에 접근하기 위한 API 서버 경로이다.

2) Deployment

1. **Backend**

spring.profiles.active=test

배포환경과 개발환경 분리를 위해 Spring Boot의 application.yml 프로필을 분리하였다. java –jar -Dspring.profiles.active=test 옵션을 통해 프로필을 활성화시킬 수 있다.

2. **Frontend**

VITE_APP_REST_API_KEY="98268e53473ceb3e11dd6e609a5fa990"

Kakao OAuth 인증을 위해 필요한 API 시크릿 키이다.

VITE_APP_IMAGE_ROOT=https://storage.googleapis.com/studymoim/

사진 파일들을 다운로드, 업로드하기 위해 접근해야하는 클라우드 파일시스템 경로이다.

VITE_APP_API_SERVER="i8a110.p.ssafy.io:8080"

Spring Boot 서버에 접근하기 위한 API 서버 경로이다.

3. **Database**

MYSQL_ROOT_PASSWORD=root

MySQL 컨테이너 최초 구동 시 root 계정의 비밀번호를 설정하기 위해 등록하였다.

MYSQL_DATABASE=studymoim

MySQL 컨테이너 최초 구동 시 기본 스키마를 생성하기 위해 등록하였다.

배포 환경

1) 컨테이너

1. **Spring Boot**

spring-dev 컨테이너를 빌드하기 위한 Dockerfile은 다음과 같다.

```
FROM ubuntu:22.04

MAINTAINER malachai<prussian1933@naver.com>
USER root
WORKDIR /home

# Java Development Kit(Zulu8) installation
RUN \
    apt update -y && \
    apt install wget vim -y && \
    wget https://cdn.azul.com/zulu/bin/zulu8.68.0.21-ca-jdk8.0.362-linux_x64.tar.gz && \
    tar -xvzf zulu8.68.0.21-ca-jdk8.0.362-linux_x64.tar.gz && \
    mkdir /usr/lib/jvm && \
    mv zulu8.68.0.21-ca-jdk8.0.362-linux_x64 /usr/lib/jvm/zulu8.68.0.21-ca-jdk8.0.362-
linux_x64
ENV JAVA_HOME=/usr/lib/jvm/zulu8.68.0.21-ca-jdk8.0.362-linux_x64
ENV PATH=$PATH:$JAVA_HOME/bin
```

코드 1. Spring Boot 컨테이너의 Dockerfile

spring-dev 컨테이너에는 JDK만 설치하여 빌드를 진행하였다.

2. **Nginx**

nginx-dev 컨테이너를 빌드하기 위한 Dockerfile은 다음과 같다.

```
FROM ubuntu:22.04

MAINTAINER malachai<prussian1933@naver.com>
USER root
WORKDIR /home

# Nginx installation
RUN \
    apt update -y && \
    apt install wget vim -y && \
    apt install nginx -y && \
    rm /etc/nginx/sites-available/default && \
    rm /etc/nginx/sites-enabled/default && \
    touch /etc/nginx/sites-available/myapp.conf && \
    ln -s /etc/nginx/sites-available/myapp.conf /etc/nginx/sites-enabled/myapp.conf
COPY nginx/sites-available/myapp.conf /etc/nginx/sites-available/myapp.conf
```

코드 2. Nginx 컨테이너의 Dockerfile

Nginx를 설치한 후, myapp.conf 설정파일을 컨테이너로 복사해주었다.

3. MySQL

mysql:8.0.32-debian 이미지를 그대로 사용하였다.

4. Jenkins

jenkins-container 컨테이너를 빌드하기 위한 Dockerfile은 다음과 같다.

```
FROM jenkins/jenkins:lts-jdk11

MAINTAINER malachai<prussian1933@naver.com>

USER root

RUN \
    apt-get update && \
    apt-get install ca-certificates curl gnupg lsb-release -y && \
    mkdir -p /etc/apt/keyrings && \
    curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o
/etc/apt/keyrings/docker.gpg && \
    echo \
      "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
      $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null && \
    apt-get update && \
```

```
    apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y

RUN \
    curl -SL https://github.com/docker/compose/releases/download/v2.15.1/docker-
compose-linux-x86_64 -o /usr/local/bin/docker-compose && \
    chmod +x /usr/local/bin/docker-compose && \
    ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

RUN \
    apt install -y curl && \
    apt install -y build-essential && \
    curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash && \
    export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" ||
printf %s "${XDG_CONFIG_HOME}/nvm")" && \
    [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" && \
    nvm install node
```

코드 3. Jenkins 컨테이너의 Dockerfile

2) 네트워크

ssafy_network 도커 네트워크를 생성하여 docker-compose로 올라간 컨테이너들을 접합시켰다. default driver를 사용하였고, 172.16.238.0/24 의 서브넷 범위를 가진다.

3) 오케스트레이션

Docker compose 구성 파일은 다음과 같다. 아래의 설정을 통해 spring-dev, nginx-dev, mysql-dev 세개의 컨테이너를 구동하여 ssafy_network 도커 네트워크에 붙인다.

```
version: "3.7"

services:
  nginx:
    container_name: nginx-dev
    build:
      context: frontend
      dockerfile: Dockerfile
    hostname: nginx-dev
    volumes:
      - type: bind
        source: /home/ubuntu/logs/nginx
        target: /home/logs
    networks:
      ssafy_network:
        ipv4_address: 172.16.238.2
    ports:
```

```
      - "80:80"
    extra_hosts:
      - "spring-dev:172.16.238.3"
      - "mysql-dev:172.16.238.4"
    stdin_open: true
    tty: true

  spring:
    container_name: spring-dev
    build:
      context: backend
      dockerfile: Dockerfile
    hostname: spring-dev
    volumes:
      - type: bind
        source: /home/ubuntu/logs/spring
        target: /home/logs
    networks:
      ssafy_network:
        ipv4_address: 172.16.238.3
    ports:
      - "8080:8080"
    extra_hosts:
      - "nginx-dev:172.16.238.2"
      - "mysql-dev:172.16.238.4"
    stdin_open: true
    tty: true

  mysql:
    image: mysql:8.0.32-debian
    container_name: mysql-dev
    hostname: mysql-dev
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: studymoim
    volumes:
      - type: bind
        source: /home/ubuntu/logs/mysql
        target: /home/logs
    networks:
      ssafy_network:
        ipv4_address: 172.16.238.4
    ports:
      - "3306:3306"
    extra_hosts:
```

```
        - "nginx-dev:172.16.238.2"
        - "spring-dev:172.16.238.3"
    stdin_open: true
    tty: true

networks:
  ssafy_network:
    ipam:
      driver: default
      config:
        - subnet: "172.16.238.0/24"
```

코드 4. 배포를 위한 docker-compose 구성 파일

4) Jenkins 설정

싱글 노드에서 Jenkins 컨테이너의 의존성을 최대한 낮추기 위해 Jenkins 컨테이너를 따로 실행하였다.

주요 계정 목록

분류	접근 방법	계정명	비밀번호
MySQL	SSH 터널링	root	SSAFYseoulNo1!
Jenkins	http://i8a110.p.ssafy.io:9090	admin	AllTime1st!

표 3. 운영 시 주요 계정 목록

2. 외부 서비스

1) 소셜 인증

1. 카카오

a. 사이트 도메인

Kakao Developers의 개발 어플리케이션에 개발, 운영 환경을 등록해야 한다. 앱 설정 > 플랫폼 > 사이트 도메인에 다음을 추가한다.

- *http://localhost:4000*
- *http://localhost:8080*
- *http://i8a110.p.ssafy.io*
- *http://i8a110.p.ssafy.io:8080*

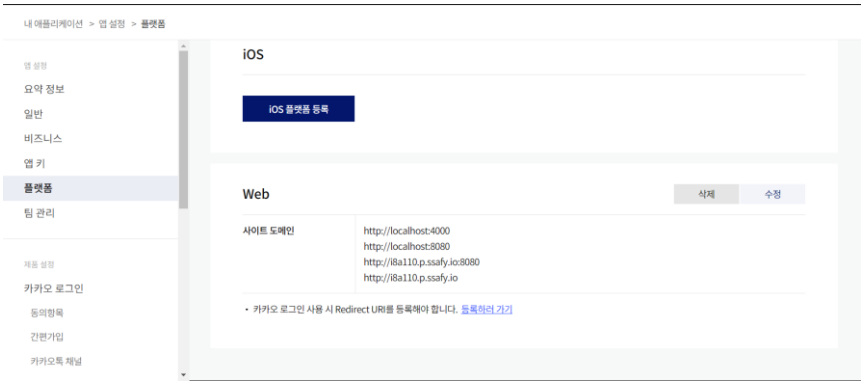

그림1. Kakao Developers 사이트 도메인 등록

b. Redirect URI

사용자 인증 후 클라이언트가 이동해야 하는 경로이다. 다음과 같이 경로를 추가한다.

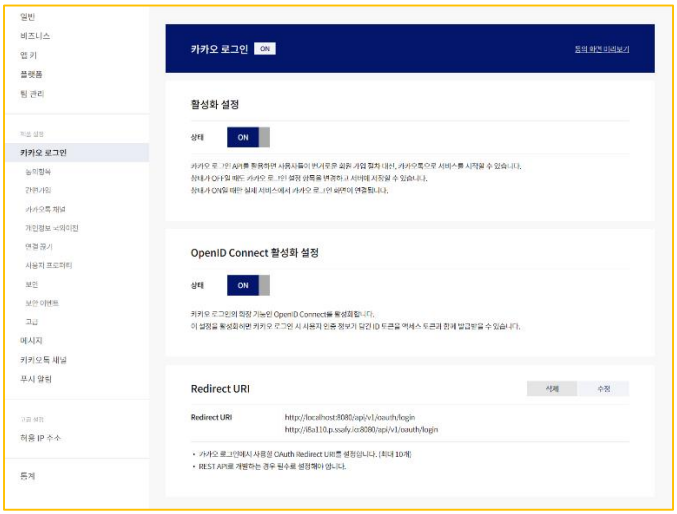

그림2. Kakao Developers 사이트 카카오 로그인 활성화 및 Redirecr URI 설정

c. 동의 항목

사용자의 이메일을 필수로 사용해야 한다. 이메일 필수 선택 외에는 사용하지 않음으로 설정한다.

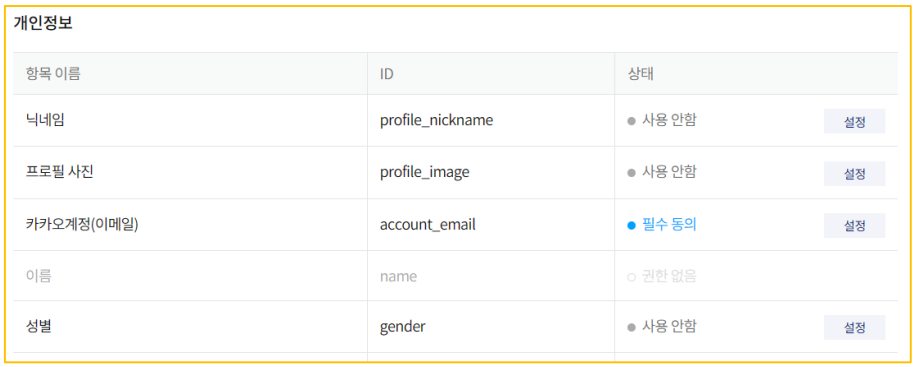

그림3. Kakao Developers 사이트 카카오 로그인 개인정보 수집 설정

2) 데이터 크롤링

1. Youtube API

https://console.developers.google.com에서 Youtube Data API v3를 사용하기 위한 키를 발급 받아야 한다. 프로젝트 등록 > API 라이브러리 > Youtube Data API v3 > 키 발급

그림 4. YouTube API 콘솔

3) 사진 파일 관리

1. GCS

a. https://cloud.google.com/storage?hl=ko 로 접속한다.

b. 콘솔로 이동하고 상단에서 새로운 프로젝트를 생성한다.

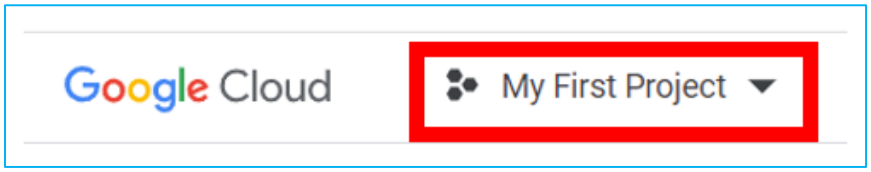

그림 5. Google Cloud에서 프로젝트 생성

c. 좌측 메뉴 > Cloud Storage > 버킷

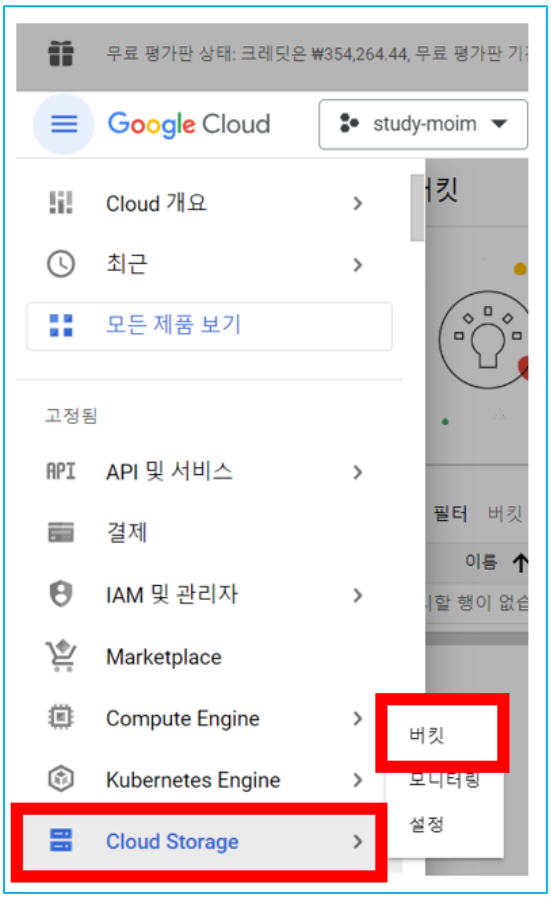

그림 6. Cloud Storage의 버킷으로 이동

d. 버킷을 새로 만든다.

그림 7. 새로운 버킷 생성

e. 버킷 세부정보 > 권한 > 엑세스 권한 부여

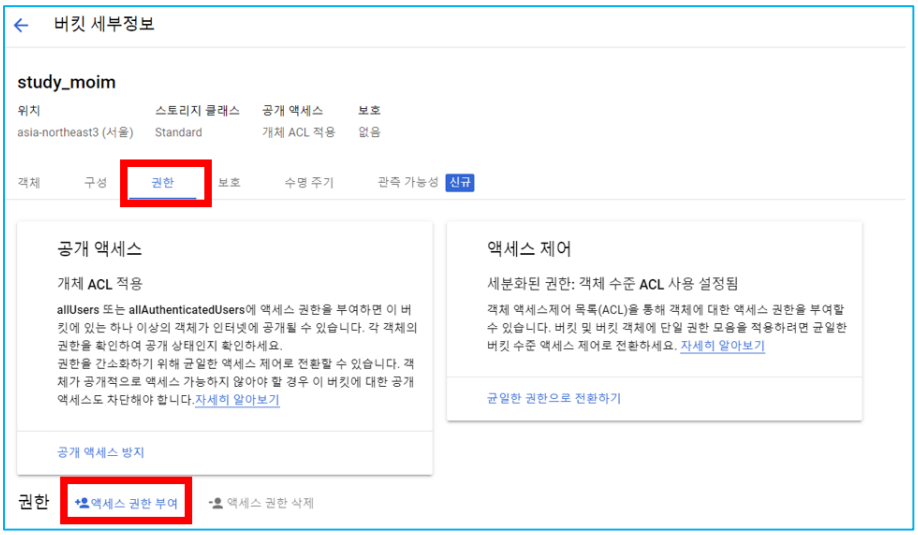

그림 8. 엑세스 권한 부여

f. allUsers에게 저장소 개체 뷰어 권한을 부여한다.

그림 9. allUsers에게 엑세스 권한 부여

g. 좌측의 메뉴 > IAM 및 관리자 > 서비스 계정

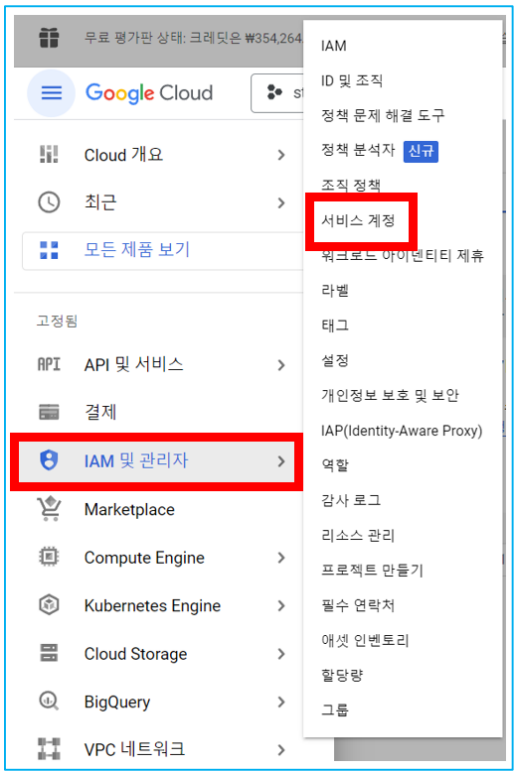

그림 10. IAM 및 관리자에서 서비스 계정으로 이동

h. 새로운 서비스 계정을 생성한다.

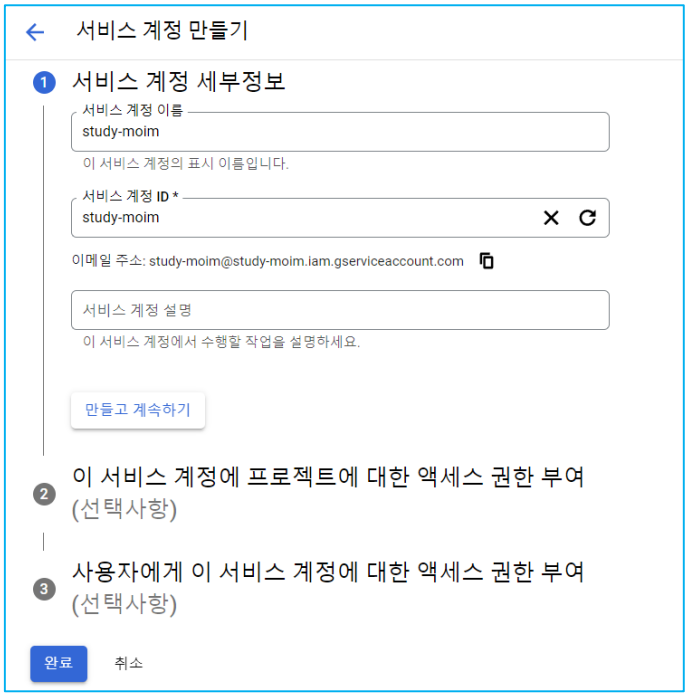

그림 11. 서비스 계정 만들기

i. 생성한 서비스 계정 > 키 > 키 추가 > 새 키 만들기 > JSON 선택 후 생성된 JSON 파일을 backend > resources 폴더 내에 추가한다.

j. application.yml에 GCS의 경로를 설정한다. 프로젝트 내에서 버킷의 이미지를 사용할 경우 https://storage.googleapis.com/study_moim/ 뒤에 파일 이름을 작성하면 원본 이미지 파일을 가져올 수 있다.

3. 배포 매뉴얼

1) 호스트 머신 세팅

1. 기본 유틸리티 설치

호스트 머신에서 사용할 기본적인 유틸리티를 설치한다.

```
sudo apt install vim -y

sudo apt install curl –y

sudo apt install net-tools -y
```

코드 5. Jenkins 컨테이너의 실행 스크립트

2. 방화벽 설정

호스트 머신에서 개방할 포트는 80, 443, 8080, 9090 포트이다. 각 포트의 사용 용도는 다음과 같다.

포트번호	용도
80	HTTP
443	HTTPS
8080	Spring Boot API Server
9090	Jenkins WEB UI

표 4. Jenkins 컨테이너의 실행 스크립트

```
sudo ufw allow 80

sudo ufw allow 443

sudo ufw allow 8080

sudo ufw allow 9090

sudo ufw enable
```

코드 6. UFW 포트 개방 명령어

3. 타임존 설정

호스트 머신의 타임존을 현지 시간으로 맞춰준다.

```
sudo ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

코드 7. 타임존 동기화 명령어

4. Docker 설치

호스트 머신에 Docker 엔진을 설치한다. 이는 Jenkins에 설치 할 Docker와 자원을 공유한다.

```
apt-get update

apt-get install ca-certificates curl gnupg lsb-release -y

mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian \
        $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

apt-get update

apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y
```

코드 8. Docker 설치 명령어

2) Jenkins 컨테이너 실행

1. Jenkins 컨테이너 빌드 및 실행

deployment/Jenkins/Dockerfile 도커 스크립트를 통해 다음과 같이 이미지를 빌드한 후 실행한다.

```
sudo docker build -t jenkins/jenkins:custom .

sudo docker run -d -it -u root -v jenkins_home:/var/jenkins_home -v
/var/run/docker.sock:/var/run/docker.sock -p 9090:8080 -p 50000:50000 --name
jenkins_container jenkins/jenkins:custom /bin/bash

sudo docker exec jenkins_container java -jar /usr/share/jenkins/jenkins.war
```

코드 4. Jenkins 컨테이너의 실행 명령어

3) Jenkins 설정

1. Jenkins 웹 UI 진입

실행 된 컨테이너는 http://[EC2 퍼블릭 도메인]:9090 으로 접근할 수 있다. 최초 실행

시 바인딩 한 Jenkins_home 디렉토리 내 서버 로그에서 어드민 키를 확인할 수 있다. 이를 통해 최초 계정을 생성, 기본 플러그인을 설치한다. 플러그인 설치 실패 시, 다음 단계에서 수동으로 설치할 수 있다.

2. Jenkins 플러그인 설치

기본적인 플러그인이 설치되지 않았을 시, 우측 상단에 Dependency 오류가 발생한다. 명시된 플러그인을 설치하면 이를 해결할 수 있다.

대시보드>Jenkins 관리>플러그인 관리 설정 메뉴를 통해 추가적인 플러그인을 설치할 수 있다. Docker 관련 플러그인과 GitLab 관련 플러그인을 설치한다.

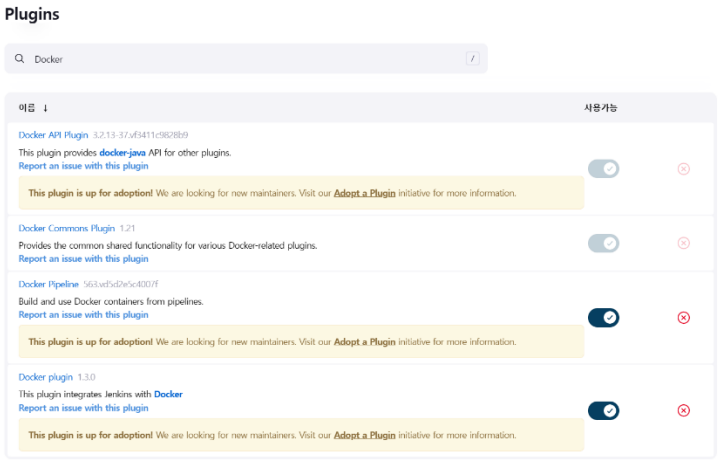

그림 12. Jenkins의 Docker 관련 플러그인

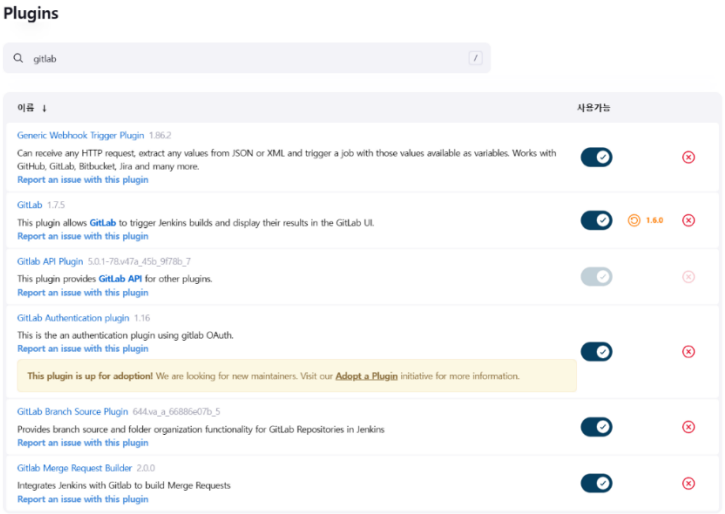

그림 13. Jenkins의 Gitlab 관련 플러그인

3. Jenkins GitLab Credentials 추가

대시보드>Jenkins 관리>Manage credentials 메뉴에서 Gitlab 연동 시의 인증 정보를 추가할 수 있다. Global 도메인에 인증 정보를 추가할 수 있다.

그림 14. Gitlab 인증 정보 추가

SSAFY의 Gitlab 인증방식은 Username with password 방식이기 때문에 알맞게 등록을 진행한다.

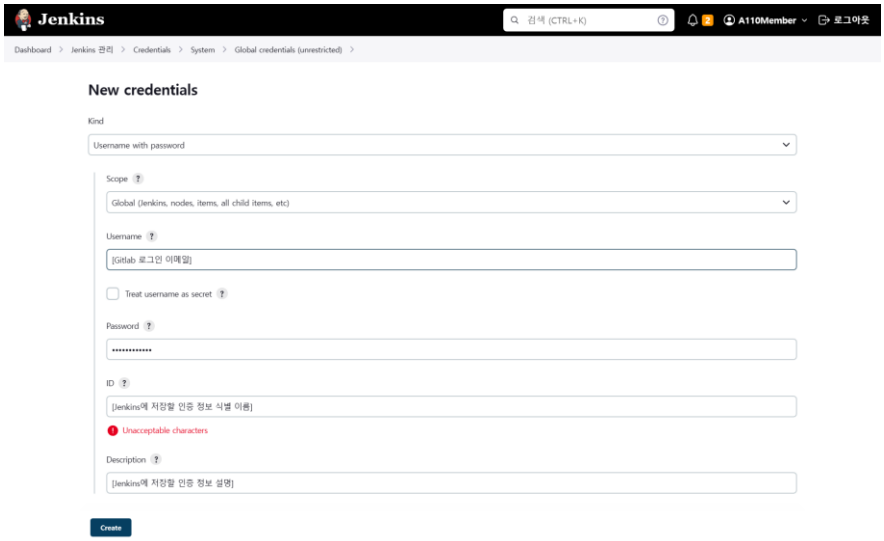

그림 15. Gitlab 인증 정보 추가

4) Jenkins 프로젝트 생성

1. 프리스타일 프로젝트 생성

대시보드>새로운 Item 메뉴에서 새로운 프리스타일 프로젝트를 생성할 수 있다.

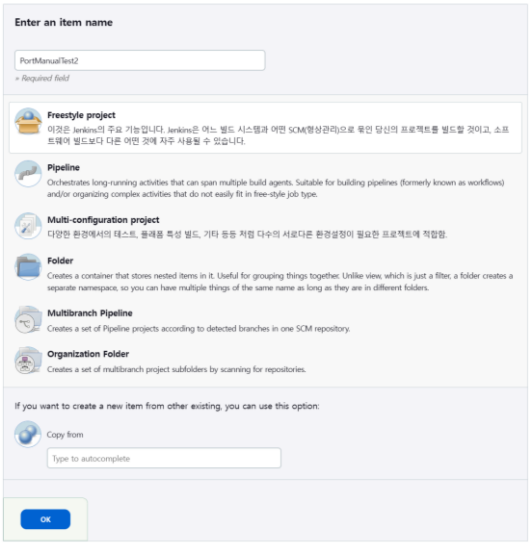

그림 16. Freestyle Project 생성

프로젝트 이름, 설명과 같은 기본적인 틀을 작성한 후 소스 코드 관리 메뉴로 이어 진행한다.

2. Gitlab 연결

Jenkins 에서 Gitlab 리포지토리를 자동으로 pull 받을 수 있도록 작업 리포지토리를 등록한다. 미리 등록한 Credential 을 통해 자동으로 접근을 확인한다.

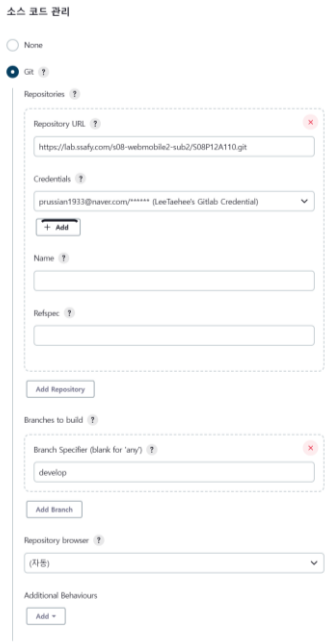

그림 17. Gitlab Repository 연결

3. Gitlab Webhook 연결

Gitlab 리포지토리에 push 이벤트가 발생하면 트리거를 전송할 수 있도록 Jenkins와 Gitlab Webhook를 연결해준다.

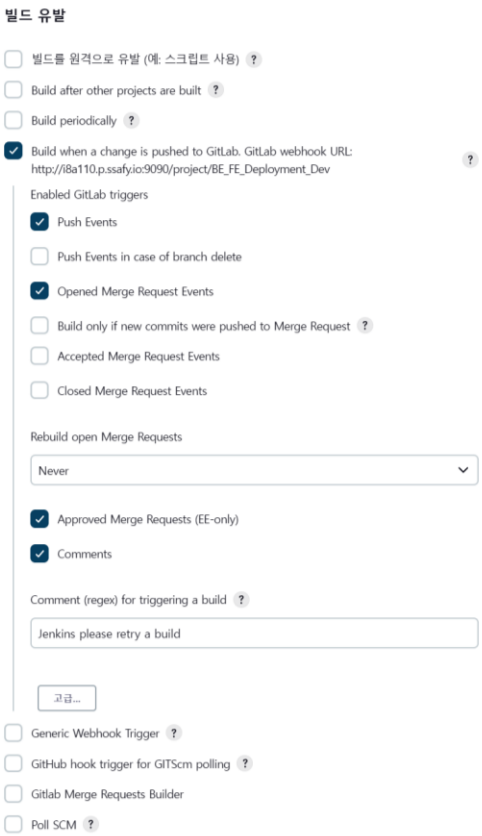

그림 18. Gitlab 이벤트 트리거 시 빌드 유발 행동 설정

Gitlab Repository의 Settings>Webhooks 메뉴에서 Webhook을 생성할 수 있다.

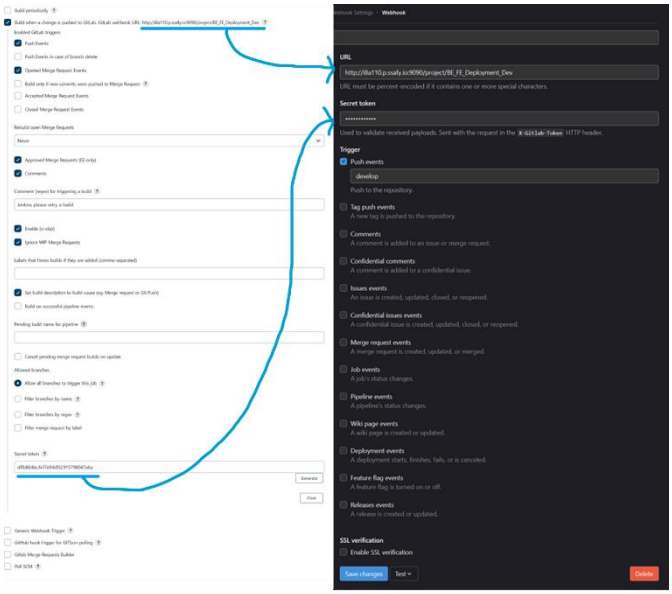

그림 19. Gitlab Webhook 생성 및 URL, Token 설정

4. Execute shell 설정

빌드 트리거 발생 시 자동으로 Spring Boot 어플리케이션과 Vite 빌드, Nginx 배포를 수행하기 위해 쉘 스크립트를 작성한다.

Spring Boot의 로그 파일은 호스트 머신의 $HOME_DIR/logs/spring/server.log 에서 조회할 수 있다.

```
cd development/Backend/studymoim
bash ./mvnw clean
bash ./mvnw compile
bash ./mvnw package
cd ../../../
docker stop spring-dev
docker rm spring-dev
docker-compose --env-file deployment/config/.env.dev -f deployment/docker-compose-dev.yml up -d
docker cp development/Backend/studymoim/target/peace-0.0.1-SNAPSHOT.jar spring-dev:/home
docker exec -d spring-dev sh -c "java -jar -Dspring.profiles.active=test /home/peace-0.0.1-SNAPSHOT.jar >> /home/logs/server.log"

cd development/FrontEnd/peace_studymoim
npm install -g npm
npm ci
npm run build
docker stop nginx-dev
docker rm nginx-dev
cd ../../../
docker-compose --env-file deployment/config/.env.dev -f deployment/docker-compose-dev.yml up -d
docker cp development/FrontEnd/peace_studymoim/dist/. nginx-dev:/home/peace_studymoim
docker exec nginx-dev service nginx start
```

코드 5. 어플리케이션 배포를 위한 스크립트

프로젝트 내의 Docker Compose 구성파일인 docker-compose-dev.yml 파일로 컨테이너의 실행을 관리하기 때문에, 최초 스크립트 실행 전 docker-compose-dev.yml을 통해 spring-dev, nginx-dev, mysql-dev 컨테이너를 구동해야 한다. 이를 위해 필요한 명령은 다음과 같다.

```
sudo docker-compose -f deployment/docker-compose-dev.yml build

sudo docker-compose --env-file deployment/config/.env.dev -f deployment/docker-compose-dev.yml up -d
```

코드 6. Jenkins의 최초 빌드 유발 전 동작되어야 하는 docker-compose 컨테이너 구성

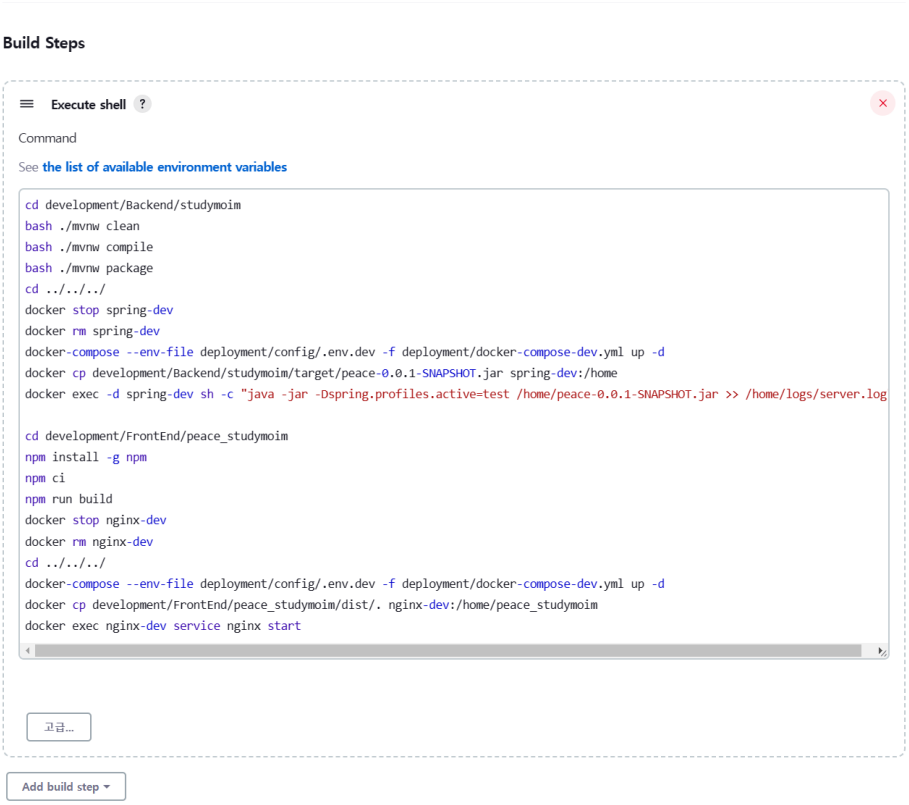

그림 20. Jenkins의 어플리케이션 배포 스크립트

빌드 수동 트리거 또는 Webhook 트리거 유발 후, 프리스타일 프로젝트의 히스토리 섹션에서 빌드 성공 유무를 확인할 수 있다.

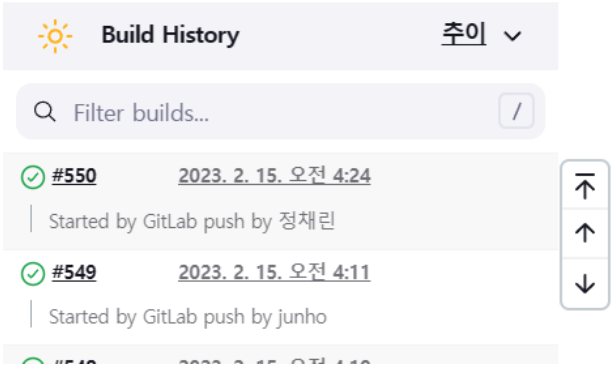

그림 21. 빌드 트리거 유발 후의 빌드 히스토리

4. 시연 시나리오

1) 메인 페이지

1. 로그인 전 메인 페이지

그림 22. 로그인 전 메인페이지

1. 메인 네비게이션
2. 로그인 버튼
3. 전체 강의 페이지 링크
4. 강의 상세 페이지 링크
5. 이전 추천 강의 보기
6. 다음 추천 강의 보기
7. 모집중인 스터디 링크
8. 스터디 상세 모집글 페이지 링크
9. 자유 게시판 페이지 링크
10. 자유 게시글 상세 글 페이지 링크
11. 강의 질문 게시판 페이지 링크

12. 강의 질문 게시판 상세 글 페이지 링크

2. 로그인 후 메인 페이지

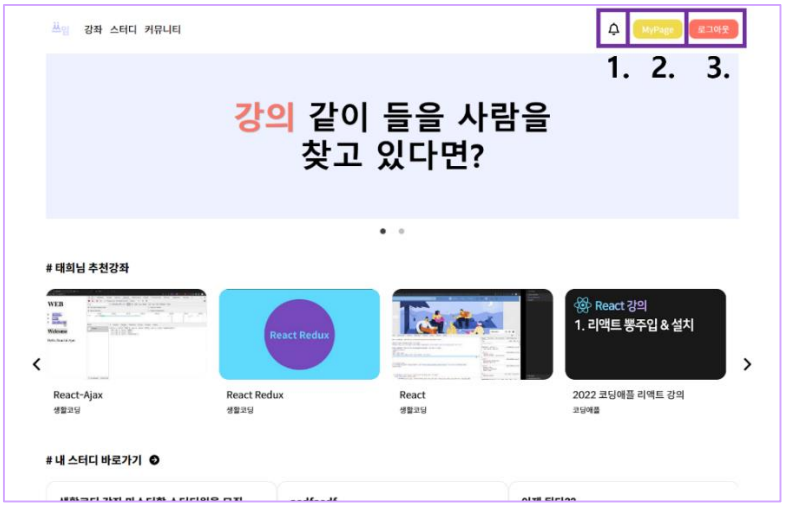

그림 23. 로그인 후 메인페이지

1. 새로 받은 알림 확인 모달
2. 마이 페이지 링크
3. 로그아웃 버튼

2) 강좌 페이지

1. 강좌 목록 페이지

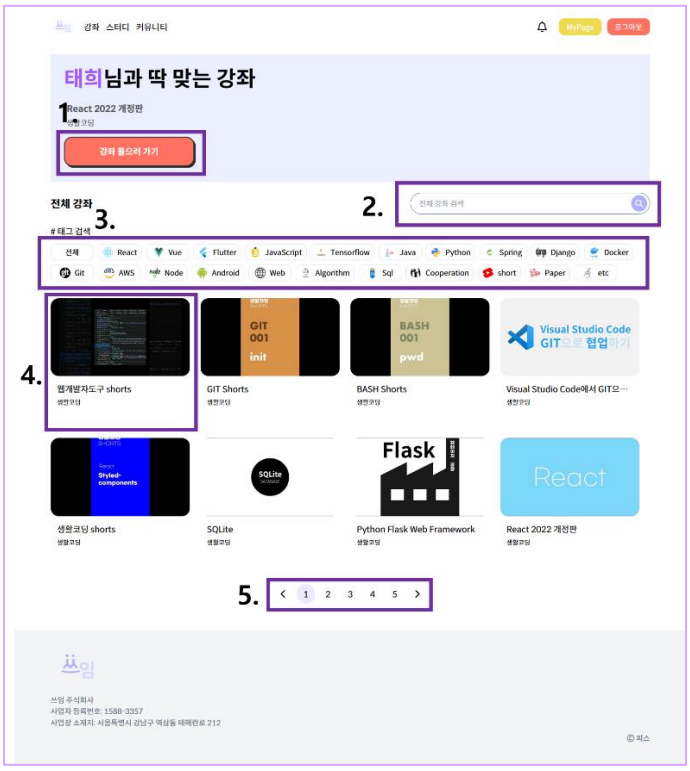

그림 24. 강좌 목록페이지

1. 추천된 강좌의 강좌 상세 페이지 링크
2. 강좌 제목 검색바
3. 강좌 태그 검색 버튼
4. 강좌 상세 페이지 링크
5. 강좌 목록 페이징 네비게이션

2. 강좌 상세 페이지

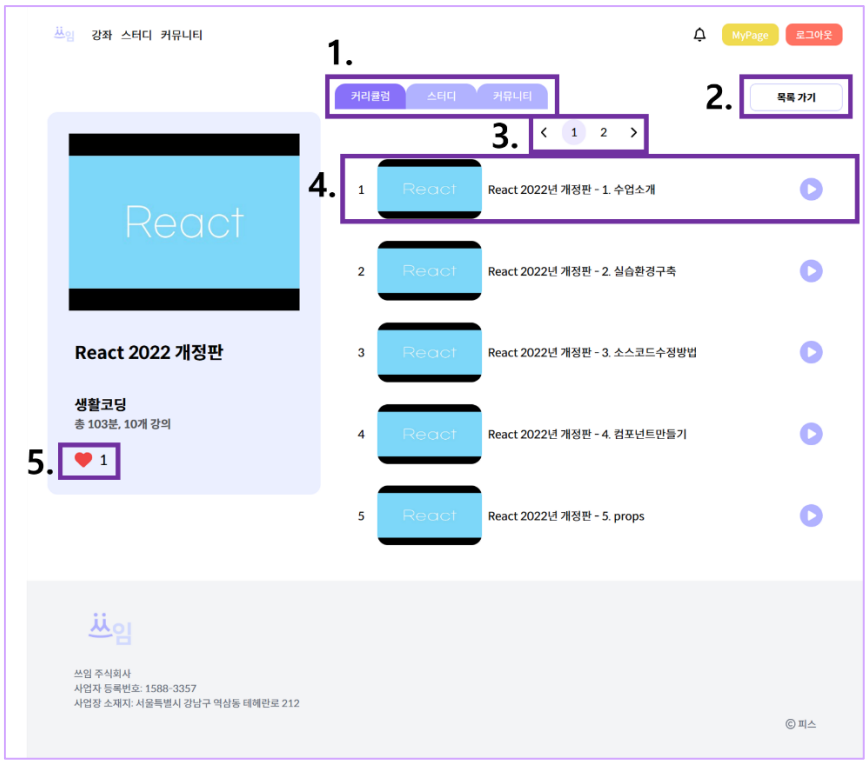

그림 25. 강좌 상세 페이지

1. 강좌를 포함하는 커리큘럼, 강좌를 포함하는 스터디, 강좌와 관련된 질문 목록 탭
2. 강좌 목록 페이지 링크
3. 강의 목록 페이징 네비게이션
4. 해당 강의 플레이어 페이지 링크
5. 강좌 좋아요 버튼

4) 플레이어 페이지

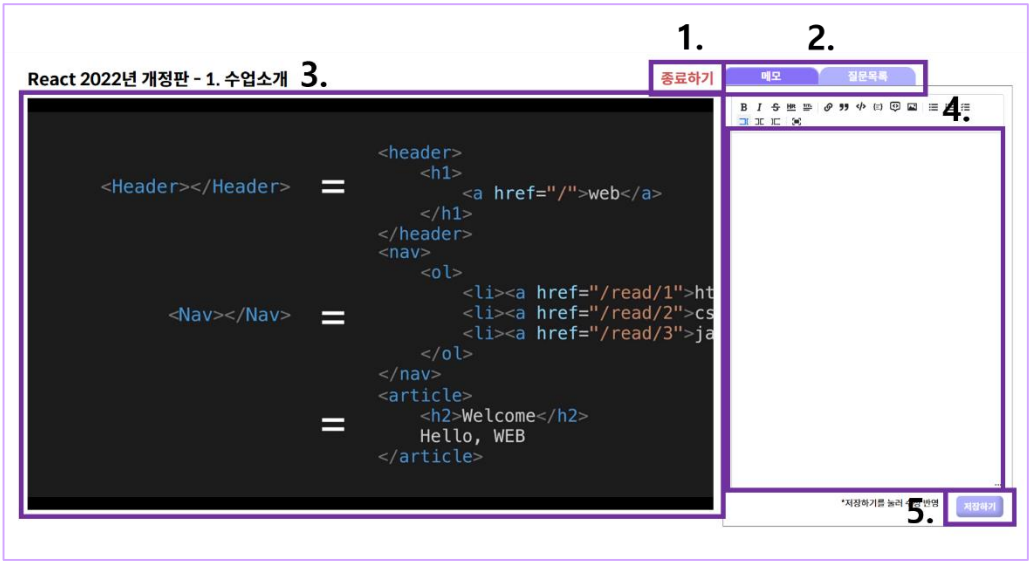

그림 26. 강의 비디오 플레이어 페이지

1. 해당 강의 플레이어 종료하기 버튼
2. 강의에 남긴 메모, 강의에 달린 질문 목록 탭
3. 강의 미디어 플레이어
4. 메모 작성란
5. 메모 저장 버튼

5) 커뮤니티 페이지

1. 게시판 목록 페이지

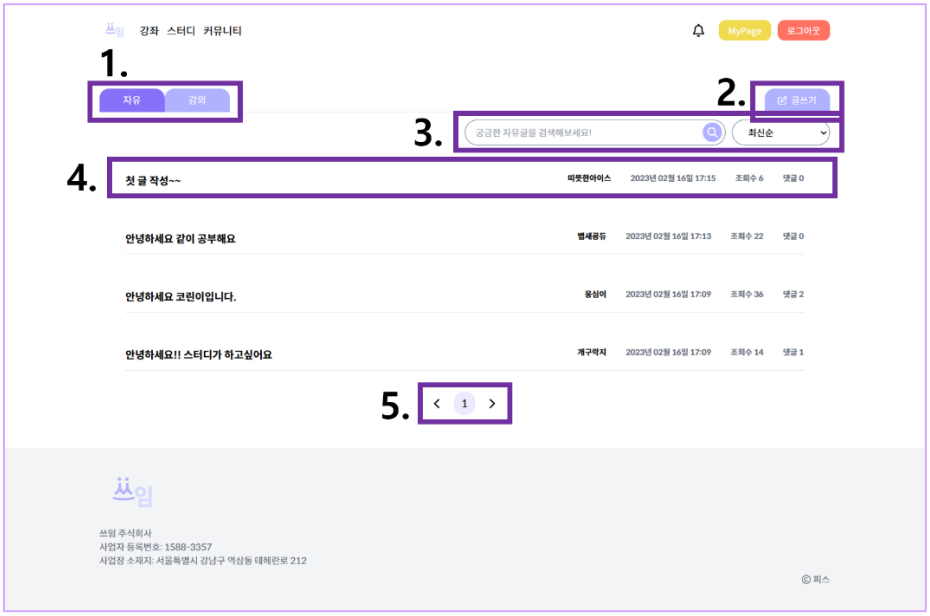

그림 27. 커뮤니티 페이지

1. 자유 게시판, 질문 게시판 전환 탭
2. 글쓰기 버튼
3. 게시글 정렬 및 검색 바
4. 해당 게시글 페이지 링크
5. 게시글 목록 페이징 네비게이션

2. 게시판 목록 페이지

그림 28. 커뮤니티 글 상세 페이지

1. 해당 유저 정보 페이지 링크
2. 해당 강의 플레이어 페이지 링크
3. 댓글 작성란

6) 스터디 페이지

1. 스터디 상세 페이지 (스터디장)

그림 29. 스터디장의 스터디 상세 페이지 실시간 강의 탭

1. 해당 유저 정보 페이지 링크
2. 해당 강의 플레이어 페이지 링크
3. 공지 작성란
4. 스터디 활동 및 관리 탭 모음

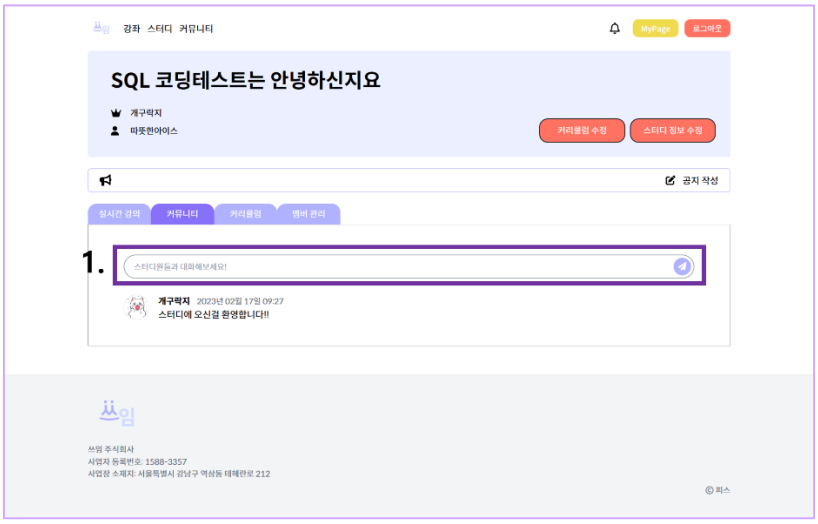

그림 30. 스터디장의 스터디 상세 페이지 커뮤니티 탭

1. 스터디 커뮤니티 글 작성란

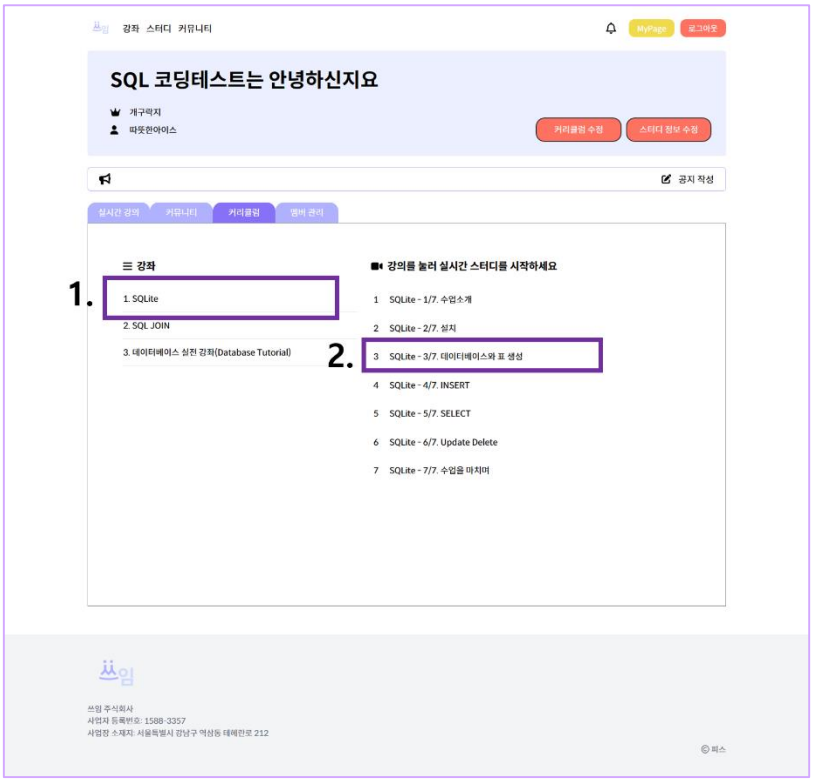

그림 31. 스터디장의 스터디 상세 페이지 커리큘럼 탭

1. 커리큘럼 강좌 목록
2. 해당 강의의 실시간 스터디 플레이어 페이지 링크

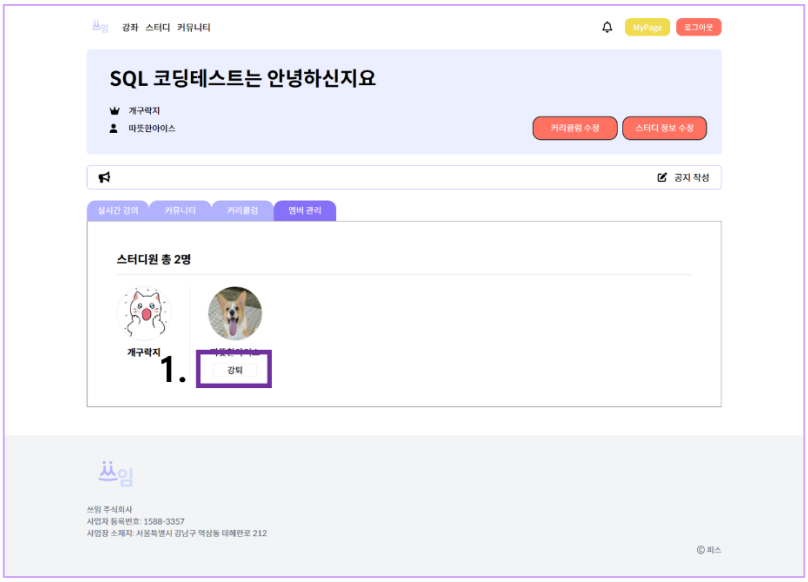

그림 32. 스터디장의 스터디 상세 페이지 멤버 관리 탭

1. 스터디원 강퇴 버튼

2. 스터디 상세 페이지 (스터디원)

그림 33. 스터디원의 스터디 상세 페이지 실시간 강의 탭

1. 스터디 활동 탭 모음
2. 진행중인 실시간 강의 시청 버튼

8) 실시간 스터디 플레이어 페이지

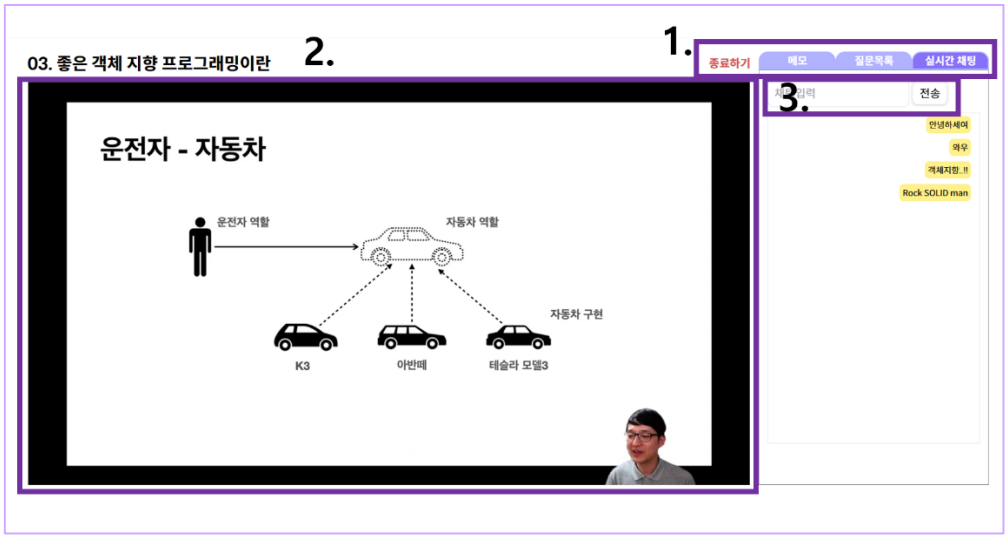

그림 34. 실시간 스터디 플레이어 페이지의 실시간 채팅 탭

1. 메모, 질문 목록, 실시간 채팅 탭
2. 시청 스터디원들과 동기화 된 강의 미디어 플레이어
3. 채팅 입력란

9) 마이 페이지

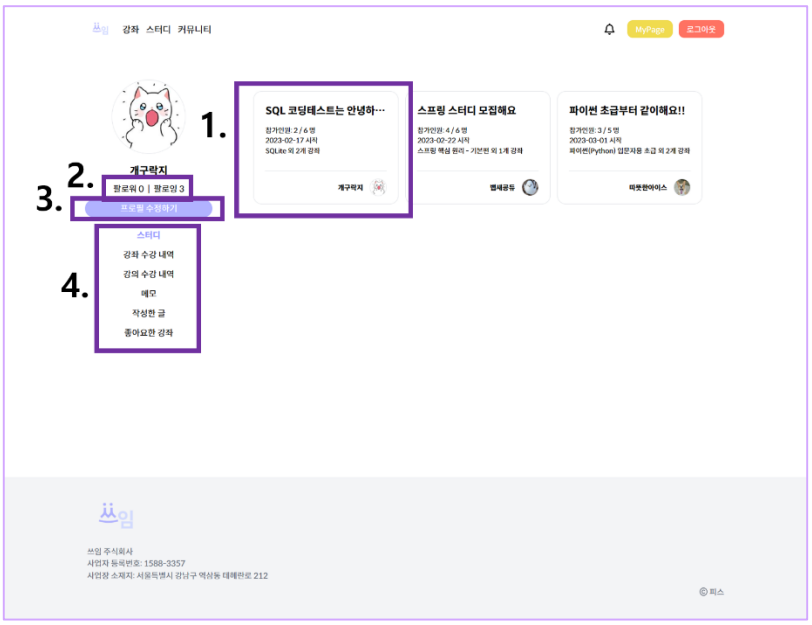

그림 35. 마이페이지 스터디 메뉴

1. 참여한 스터디 상세 페이지 링크
2. 팔로잉, 팔로워 목록 확인 버튼
3. 내 정보 수정 버튼
4. 마이페이지 각종 정보 모음

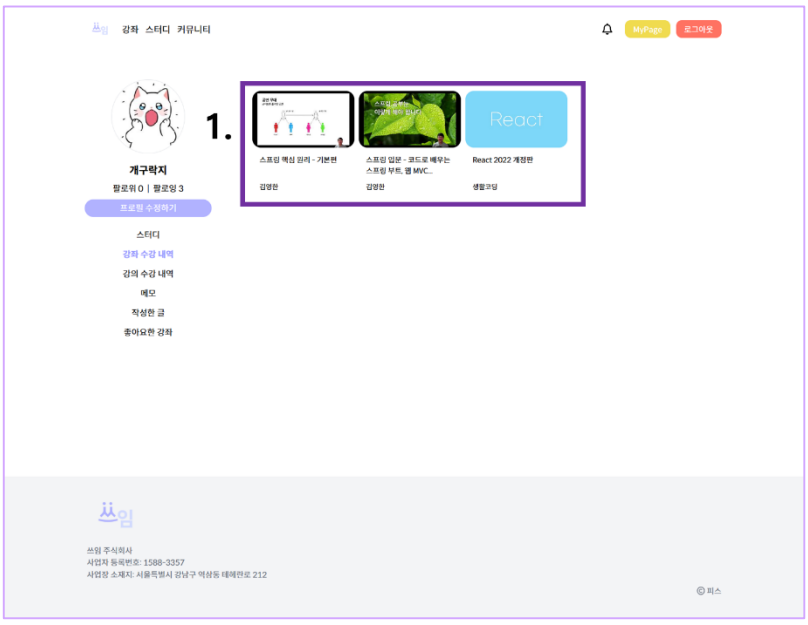

그림 36. 마이페이지 강좌 수강 내역 메뉴

1. 수강한 강좌의 상세 페이지 링크

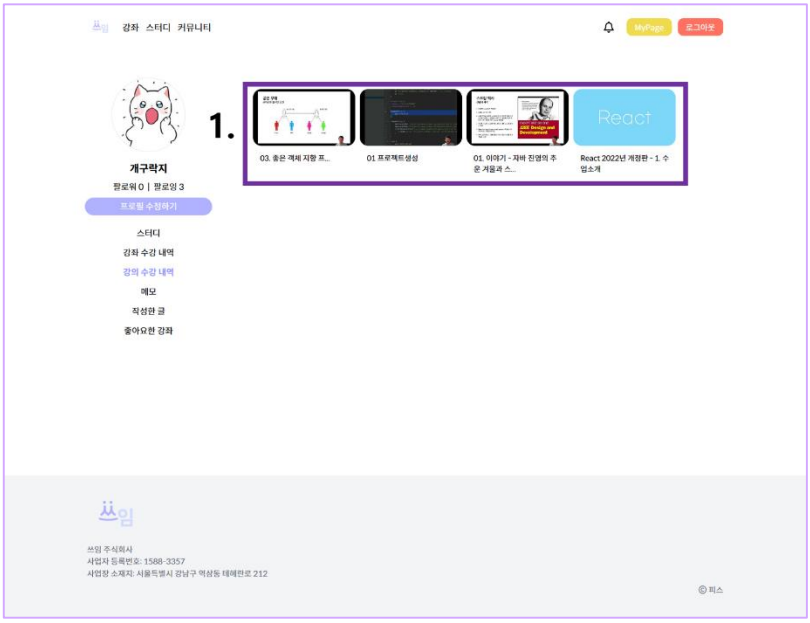

그림 37. 마이페이지 강의 수강 내역 메뉴

1. 수강한 강의 플레이어 페이지 링크

그림 38. 마이페이지 메모 메뉴

1. 강의 별 저장한 메모 목록

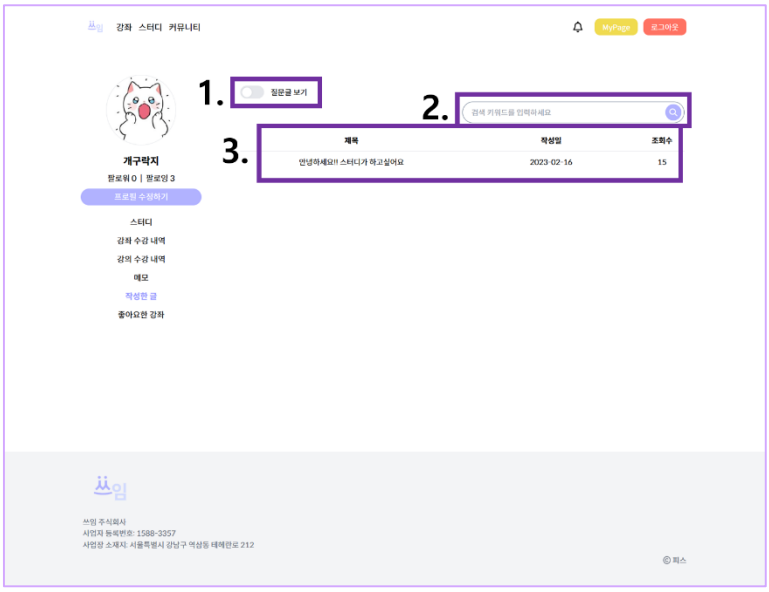

그림 39. 마이페이지 작성한 글 메뉴

1. 작성한 글 분류(자유/질문) 토글 버튼
2. 작성한 글 검색 바
3. 작성한 글 상세 페이지 링크

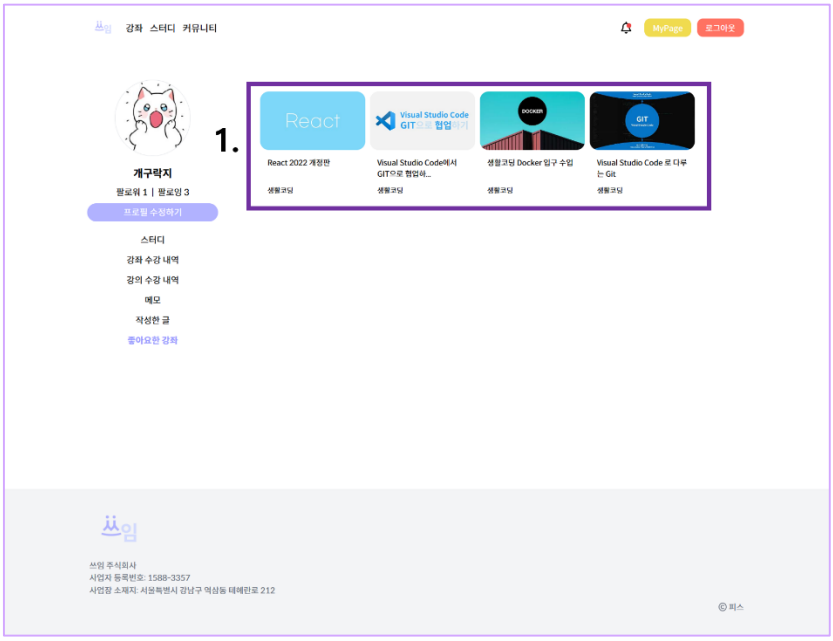

그림 40. 마이페이지 좋아요한 강좌 메뉴

1. 좋아요 표시한 강좌 목록 및 해당 강좌 상세 페이지 링크